Collison Avoidance using Point Cloud Data Fusion from Multiple Depth Sensors: a Practical Approach

Matteo Melchiorre
Department of Mechanical and Aerospace Engineering DIMEAS
Politecnico di Torino
Torino, Italy
matteo.melchiorre@polito.it

Leonardo Sabatino Scimmi
Department of Mechanical and Aerospace Engineering DIMEAS
Politecnico di Torino
Torino, Italy
leonardo.scimmi@polito.it

Stefano Paolo Pastorelli
Department of Mechanical and Aerospace Engineering DIMEAS
Politecnico di Torino
Torino, Italy
stefano.pastorelli@polito.it

Stefano Mauro
Department of Mechanical and Aerospace Engineering DIMEAS
Politecnico di Torino
Torino, Italy
stefano.mauro@polito.it

***Abstract*— This paper presents a collision avoidance system based on vision sensors that is suitable for collaborative robotics scenarios. In fact, collaborative robotics foresees the possibility that humans and robots share the same workspace, so the safety of the human operators must be ensured. The collision avoidance algorithm here presented can modify the trajectory of the robot in order to avoid any collisions with a human operator. A fundamental element for the algorithm is the relative position between robot and human. In this work, the information of the position of a human operator is obtained by Microsoft Kinect sensor in the form of a point cloud. Two Microsoft Kinect are used and their point cloud data is merged to overcome the problems related to the possible occlusions of the sensors, obtaining a more reliable point cloud. Each Kinect works with a dedicated PC and the two PCs communicate via ethernet network in a master-slave mode. The layout of the acquiring system is described and the functions used for the communication between the PCs and the manipulation of the point clouds are presented. Results of simulation tests made to verify the performances of the system and collision avoidance based on point cloud compared with convex mesh are reported and discussed.**

Keywords—collaborative robotics, collision avoidance, trajectory planning, vision system, depth sensors, point cloud

I. INTRODUCTION

Collaborative robotics represents today one of the most fascinating field of industrial robotics. The introduction of a new class of robot, named "Cobot", designed to collaborate with the human, opens new scenarios in which the human and the robot help each other in order to optimally achieve a task. The idea is to combine the flexibility and the supervision of the human operator with the accuracy, repeatability and resilience of the Cobots. This would limit the fatigue and the stress of the worker, increasing the quality of the output product. The most common industrial applications distinguish the cases where it is required that human and robot meet to complete the task from the operations in which the contact must be avoided. The first can be an assembly task with the possibility to jointly handle objects [1]; the latter is the case of a sequential task, in which there is not cooperation at the level of hand-over parts [2]. To make those possible, the robot and the human should move synchronously, it means they must recognize each other and act to efficiently and safely accomplish the task. Thus, the interface elements and the control strategies assume a crucial role and should be characterized depending on the task.

As regards the interfaces, vision-based approach such as 3D surveillance by means of depth cameras constitutes the most credible alternative to wearable tracking systems, which would limit the human operator with suits, sensors or markers to track his motion. Moreover, the release of cheap and powerful devices like Microsoft Kinect, made the human tracking accessible, introducing robust and fast skeleton tracking by means of an IR sensor [3] and dedicated SDK [4]. Even if depth cameras are widely used, they still have some limitations that affect their actual usage in collaborative applications, where occlusions due to the presence of objects or the robot itself can lead to errors in human tracking, compromising safety. To face this issue the use of multiple sensors is a necessary but not sufficient condition, since it translates into data fusion problem. In some cases, the merging of the raw data (point cloud) is codified to extract the human skeleton [5], while other techniques propose a more practical approach combining the information coming from two skeletons to better estimate human joints position [6]. Even if the cited methods show concrete improvements, they stand on algorithms to interpret the depth information and thus suffer the limitations of registration methods [7]. A solution consists in directly using the point cloud to identify the human body volume or the position of dynamic objects: the errors can only lead to a loss of information, e.g. due to occlusions or reflective materials [8], not to a wrong human or object pose estimate. This is crucial because it is simpler to handle the lack of data than to fix the misleading one. On the other hand, obtaining a correct and fast fusion of the depth information coming from multiple sensors becomes more challenging. Therefore, the problem transfers into sensor alignment and time efficient process of large amount of data.

The signals coming from sensors are used to apply control strategies on the robot. When finding a collision free path from the starting configuration to the goal position is required, it is common to refer to global strategies (typical method for mobile robot, also called "planning algorithm"), while local strategies are based on potential fields and generate force or velocity input to the robot [9]. Local strategies are preferred when moving in dynamic environments, since fast algorithms are needed to update the robot trajectory. The most of collaborative robotics applications use local strategies to drive the robot end-effector in the manipulator operational space. If depth cameras are used as interfaces, they can feedback depth data up to 30 Hz [4], so computing the relative human-robot position and executing the control algorithm should be done below 33 ms. Moreover, the

978-1-7281-3998-2/19/$31.00 ©2019 IEEE

robot controller usually works at higher frequency, so it would be useful to obtain input data as soon as possible. This exploits the link between the tracking strategies efficiency and the control one.

This work aims to propose and evaluate a collaborative practical setup which allows a 6dofs serial manipulator to avoid the human, tracked by directly merging point cloud from multiple depth cameras. It is straightforward to figure out that setting up a collaborative environment requires adequate hardware and good knowledge in programming to obtain high performance. These ingredients are even more essential when data fusion is chosen to improve the interface, so that measurements must be collected and processed before to be sent to the robot control algorithm. To represent and process the depth information coming from the sensors, different contributes can be found in literature. [10] proposes a first approach with multiple depth sensors, but only the minimum distance between the human convex hull and the robot is detected. [11] presents a real-time collision avoidance with potential field based on 3D point cloud data processed by GPU; in this work the GPU significantly contributes to speed up calculations, but only one sensor is used and the volumes are represented as voxels, it means that the original measurements from the sensor are approximated, with a loss of information and resolution. In [12] the human point cloud from two Kinect is represented using bounding boxes and interfaced to a virtual robot to perform collision avoidance only on TCP; thus, the signal is approximated and calculation is made considering only the minimum distance between the human volume and the end-effector. [13] introduces a depth space approach to quickly obtain distances between dynamic obstacles and a manipulator; the results are significant, but algorithm stands on a depth grid which is fixed and even if it can collect depth measurements from a second sensor to fill occlusions, it cannot benefit from the increase of resolution deriving from data merging. [14] utilizes a layout with two depth cameras and a software implementation similar to [15] to calculate distance between point cloud and a 6dofs robot, but few details about the hardware architecture and the times to compute distances are described.

For this work purpose, two Microsoft Kinect v2 connected to two PC running Matlab with a master-slave architecture are utilized. The point cloud data processing is partially parallelized between the two computers and then collected on the master to merge the measurements. Thus, distances between 9 control point on a simulated 6dofs Cobot and the human point cloud are calculated to perform a collision avoidance algorithm based on previous works [16,17,18]. The results compare the outcome in terms of calculation times and trajectories of the end effector if either a pure point cloud or a convex hull are considered. The major contribute is the proposal of an easy and reproducible layout, with a simple software usage and whose control algorithm runs below the Kinect refresh rate.

II. Layout of the Acquiring System

The hardware architecture layout is showed in Fig.1. Two Microsoft Kinect are connected to two computers running Windows 10, equipped with i7-6700 processor and 32 GB RAM.

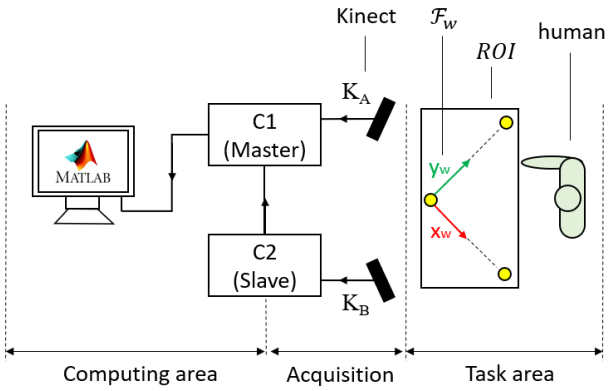

Figure 1. Hardware architecture for data acquisition and processing.

Since all calculations are made in Matlab and the Kinect SDK supports only one sensor at a time, each Kinect must be connected to a dedicated computer to properly acquire data [4]. Moreover, because the point cloud processing is time consuming, each computer can be used to handle and tune the point cluster coming from the relative sensor. This represents an important point to speed up data processing. The two machines are wired with ethernet cable to exchange data in a hybrid master-slave configuration: they acquire point clouds from the Kinect and both perform a first point cloud tuning, before that slave (C2) sends data to master (C1). The point cloud merging and the distance calculation are then executed by the master, which also runs serially the collision avoidance algorithm and the robot simulation. In Fig. 1 even the reference points (yellow dots) for cameras spatial matching and the world frame F_w are represented. To focus on the task area, the 512x424 Kinect depth image is finally reduced to a region of interest (ROI), which consists of robot operational space and workbench. The bottleneck of the proposed architecture is the point cloud data transferring from slave to master. Matlab built in communication protocol objects are not suited for fast data exchange, since during experimental tests it took more than 20 ms to transfer the point cloud data from C2 to C1. Thus a custom TCP/IP Matlab Java Interface is used to send and receive TCP packets [19]. With the helper java class option [20] the two computers are able to write and read point cloud data with an average value of 5 ms, which is enough for this application. To synchronize the two sensors, C1 triggers C2 through a dedicated port.

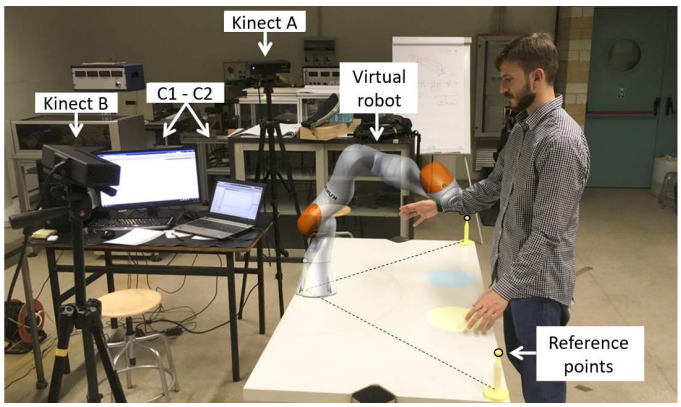

Figure 2. Laboratory set up.

The laboratory setup is presented in Fig.2. The 2 Kinect sensors are displaced at a distance of 2 m from the human, so that their field of view covers the entire workbench from different angles. The human moves handling objects on the table as it would perform an assembly task, while the simulated robot is moving with its base frame coinciding with the reference frame. Cameras spatial matching is performed using the reference points as described in [1] with an average accuracy error < 2 mm, which is the value obtained from the accuracy error distribution map for Kinect discussed in [8].

III. Point Cloud Data Processing

The measurements are acquired from the two depth sensors in a 2.5D format. It means, for each camera pixel, identified by coordinates u and v in the pixel frame, the component z_k of the observed point in Kinect frame $\boldsymbol{p}^k = (x_k\ y_k\ z_k)^T$ is measured (Fig. 3). To transform from pixel coordinate system to world frame, a pinhole camera model with radial distortion is used [21], giving the following relationships:

$$\boldsymbol{A}_w^k = (\boldsymbol{R}|\boldsymbol{t}) \qquad \boldsymbol{K} = \begin{pmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix} \tag{1}$$

$$\boldsymbol{p}^k = \boldsymbol{A}_w^k \boldsymbol{p}^w \qquad \boldsymbol{p}' = \boldsymbol{p}^k / z_k \tag{2}$$

$$\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \boldsymbol{K}\boldsymbol{p}' \tag{3}$$

$$x_d = x \cdot (1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \tag{4}$$

$$y_d = y \cdot (1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \tag{5}$$

$$r = \sqrt{x^2 + y^2} \tag{6}$$

were $\boldsymbol{R}$ is the rotation matrix from Kinect frame to world frame, $\boldsymbol{t}$ is the translation vector, $f_x f_y$ are the focal lengths, $c_x\ c_y$ are the coordinates of the principal point in the image plane and $k_1\ k_2\ k_3$ are the coefficients of radial distortion. $\boldsymbol{K}$ defines the intrinsic parameters of the camera, while $\boldsymbol{A}^k_w$ is the extrinsic matrix. If $\boldsymbol{K}$ is unknown, calibration procedure is required for each camera. In this case $\boldsymbol{K}$ is obtained directly from the camera using the C++ wrapper function in [22], passing through Microsoft SDK. Table 1 summarizes the intrinsic parameters of the two Microsoft Kinect.

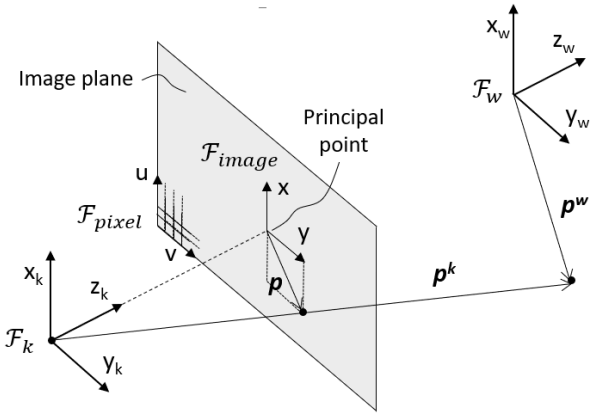

Figure 3. Pinhole camera model.

Table 1. Kinect intrinsic parameters

(values in pixels)	Kinect A	Kinect B
f_x	366.0512	367.4633
f_y	366.0512	367.4633
c_x	260.5019	256.212
c_y	205.2539	208.8752
k_1	205.2539	208.8752
k_2	-0.27082	-0.2719
k_3	0.098178	0.10081

Using equations (1)-(6) and values in Table 1 it is possible to convert points from depth space to cartesian space, overcoming the "pcfromkinect" built in function of the Matlab Image Acquisition Toolbox, which is too slow for this application. For instance, the conversion using a custom function and the formula (1)-(6) is up to five time faster than the "pcfromkinect" function. Once the point clouds in the world frame are available on both C1 and C2, the human volume is extracted by means of segmentation. Thus, the signal is downsampled and denoised using the Computer Vision System Toolbox. This step is required to speed up calculation and to eliminate outliers, which would lead to wrong distance estimation for the collision avoidance algorithm. Successively, the tuned point clouds are collected and merged in C1 to obtain the human shape.

Fig. 4 illustrates the steps for point cloud processing. In this example, a grid step value of 0.01 is chosen for the downsample Matlab function and the merged human point cloud consists of 11372 points in the cartesian space. Notice how the use of two sensors fills the self-occlusion of the human. In Fig. 5 the effect of different downsampling grid steps is shown.

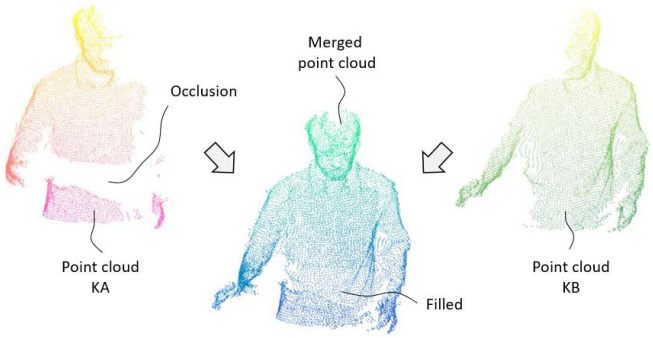

Figure 4. Point cloud merging from Kinect A and B.

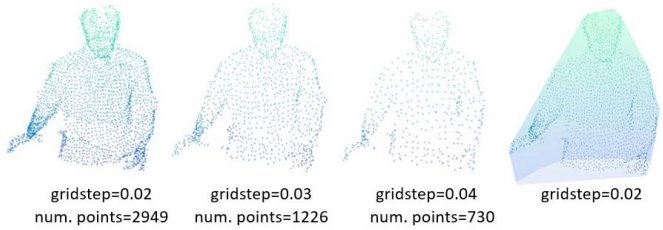

Figure 5. Point clouds and convex mesh with different downsampling.

The convex hull is built from the grid 0.02 point cloud. Moreover, it was observed that using different grid steps does not produces significant differences in triangulation, since the mesh is convex. To interface the point cloud with the robot, two different techniques are used. The first one simply considers the point cloud and the minimum distance between all the human points and a discrete number of control points on the robot is calculated using k-nearest neighbors (knn) method. The second approach is based on the convex hull of the point cloud generated by means of triangulation; here the minimum distances are estimated using mesh to point algorithm [23]. The second approach excludes the case in which the robot would choose a path between the human hand and his body, representing a more conservative approach.

IV. Robot Control Strategy

A serial manipulator with 6dofs is considered, whose kinematics is modelled as Kuka LBRiiwa14-820, bypassing its 3rd joint. A local control strategy based on potential field is adopted to move the robot from a starting point to an ending point on the workbench, avoiding collision with the detected human. The algorithm drives the robot computing repulsive velocities with the model proposed in [24] and extended in [18]. In particular, 9 control points are chosen to represent the robot volume by means of spheres (Fig. 6). Thus, the control points are grouped in 3 sets, named L1, L2 and L3. For each set, the minimum distance from the human is considered and the contribute of the repulsive velocity is added to the respective control point. If $\boldsymbol{d}_{\mathrm{Li}}$ identify the minimum distance vector between the point cloud and the robot set Li, the equation of velocity for i=1,2,3 is

$$v_{Li} = \frac{v_{max}}{(1 + e^{(\|\boldsymbol{d}_{Li}\|(2/\rho)-1)\alpha})} \tag{7}$$

$$\boldsymbol{v}_{rep\,,Li} = v_{Li} \frac{\boldsymbol{d}_{Li}}{\|\boldsymbol{d}_{Li}\|} \tag{8}$$

where $v_{\max}$ is the maximum velocity value and ρ=0.1, α=9 are the repulsive velocity parameters used for simulations. To convert the cartesian velocity to velocities in the joint space, the partial Jacobian related to the point of Li of minimum distance is calculated:

$$\dot{\boldsymbol{q}}_{Li} = J_{Li}^{-1} \cdot \boldsymbol{v}_{rep\,,Li} \tag{9}$$

The effect of the repulsive velocity of each set is added to modify the planned velocity $\boldsymbol{v}_{\mathrm{PL}}$, allowing the robot to avoid collision with obstacles with all its links:

$$\dot{\boldsymbol{q}} = J^{-1} \cdot (\boldsymbol{v}_{PL} + K\boldsymbol{e} + \boldsymbol{v}_{rep\,,L3}) \tag{10}$$

$$\dot{\boldsymbol{q}}(1{:}3) = \dot{\boldsymbol{q}}(1{:}3) + \dot{\boldsymbol{q}}_{L2} \tag{11}$$

$$\dot{\boldsymbol{q}}(1{:}2) = \dot{\boldsymbol{q}}(1{:}2) + \dot{\boldsymbol{q}}_{L1} \tag{12}$$

The velocities in (10) permit to avoid collisions between the end-effector and the human while pointing the robot task target, instead equations (11)-(12) take account of the collision avoidance between L2-L3 and the human. Finally, joints positions are computed by means of first order integration. The dynamic behaviour of the robot is also considered by modelling the velocity response of each servo axis as first order system.

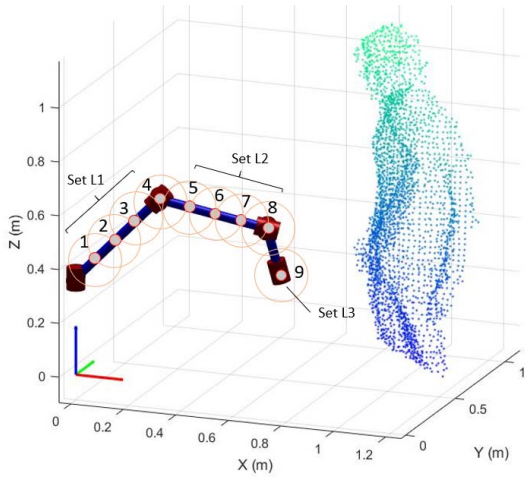

Figure 6. Matlab simulation environment built with Corke toolbox [25].

V. Results

A. Calculation times

Table 2 shows the mean values of calculation times considering 300 test samples (equivalent to 10 s) for different downsampling grid steps (decreasing the "grid" value will result in higher resolution of the point cloud). The standard deviation for each measurement was close to zero, so it is omitted. The knn-search Matlab function was used to calculate distances between the point cloud and the 9 control points on the robot, while for the convex hull the function [23] gives the minimum distance between the triangulation and each of 9 points of the robot. In all cases the entire algorithm performs at higher frequency than Kinect sensor refresh rate. This is a significant result since Matlab is not optimized for real time application with large amount of data. Notice that the Kinect segmentation of the human body was utilized to extract the human shape from the entire scene, even if it could be an issue if a real robot is utilized instead of a virtual one: in that case there is the possibility that the robot covers both Kinect for some value of joint position vector $\boldsymbol{q}$, resulting in a noisy human point cloud boundary. To overcome this limit, the removal of the point cloud portion occupied by the robot can be performed using [26]. Another possibility is to use more than two sensors, but in that case the processing times would increase. Fig. 7 represents a comparison of distance calculation in the starting frame of the test, in which the human is going to move his hands towards the robot. The convex hull approach is more conservative as expected since the robot is detected at a lower distance from the mesh. Observing the calculation times during experiments, building the triangle mesh and calculate distances, or computing distances with knn, has the same weight in terms of time processing. It means both point cloud and convex hull can be chosen to detect the human motion if fast data flow is required.

B. Collision avoidance driven by point cloud

In Fig. 8 the path of the end-effector for both collision avoidance with point cloud and convex hull are compared. The test is carried out with a grid step of 0.02. The human reaches the workbench with his right arm, while the serial manipulator is pointing towards the final task configuration. When the hand

crosses the robot planned trajectory, the end-effector starts to move away from the obstacle, choosing a path which depends on the shape of the obstacle and its relative position from the human operator. If the latter is represented by a triangle mesh, the convex hull fills the space between the human hand and his body (Fig. 7). The resulting distance vectors from the control points on the robot are different both in module and direction. Focusing on the end effector, this can be observed also in Fig. 9 (a), where the normalized repulsive velocity vectors of the end effector $\boldsymbol{v}_{rep,L3}/\, v_{max}$ is plotted for each test sample. The blue vectors refer to the point cloud approach while the red ones to the convex hull. When the obstacle is represented by means of point cloud, the robot drives its end-effector choosing to pass above the hand, in a resulting path which is closer to the human body. On the other hand, the mesh generates a repulsive velocity vector which deforms the path backward. Therefore, when safety is a crucial factor, it can be useful to compute distances from a convex hull rather than a point cloud. The influence of downsampling on the end-effector paths when the point cloud is used to compute distances is presented in Fig.9 (b). The curves are very similar in all cases, which means that the grid step does not significantly influence the robot trajectory. This suggests to select the larger grid step in order to reduce the computation time. Fig. 10, finally, describes the effect of the grid step (indicated with different line types) on computed distance values for each link set (drawn with different colors). In particular, the minimum distance from each link set is plotted as a function of the task time. The distance values are similar in all cases, which means that downsampling does not significantly influence the relative position between the human body and each of the robot links.

TABLE 2. CALCULATION TIMES

	Point cloud (knn)			ConvHull
(times in ms)	grid 0.02	grid 0.03	grid 0.04	grid 0.03
Point c. extract	2.3	2.3	2.3	2.3
Downsamp.	4.1	4.1	3.8	4.1
Denoising	8.3	6.1	4.3	6.1
Send data	5	5	5	5
Merging	2.1	2	1.7	1.9
Dist. Cal.	2.5	1.4	1	2.5
C.Avoid.	5	5	5	5
Tot.	29.3	25.9	23.1	26.9

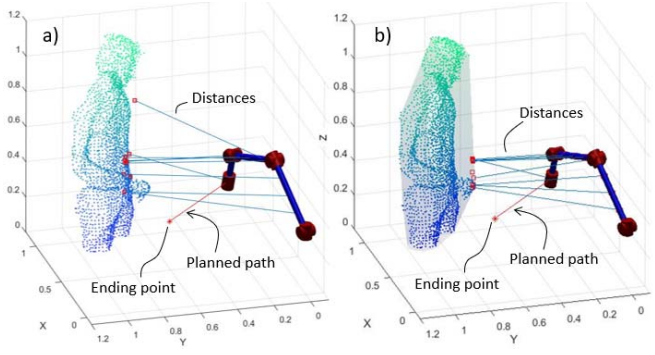

Figure 7. Point cloud merging and distance calculation by means of point cloud (a) and mesh (b) approaches.

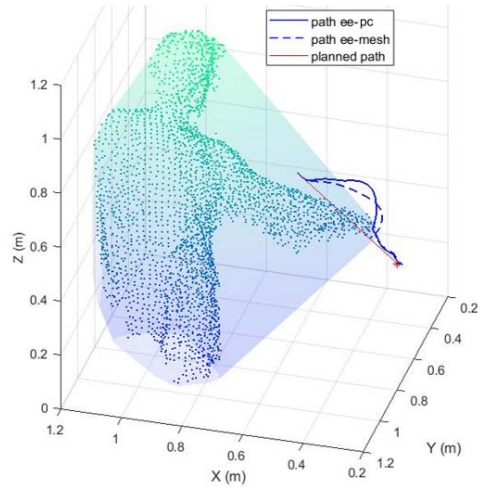

Figure 8. Collision free paths of the end-effector (ee).

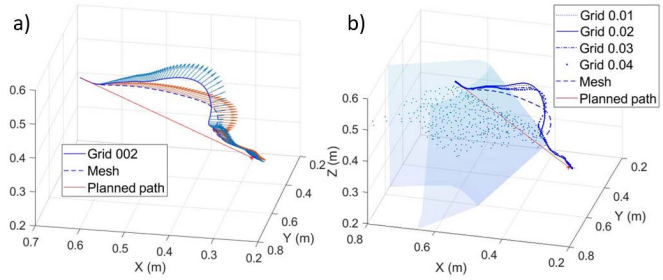

Figure 9. Contribute of the repulsive velocity (a); influence of different downsampling on end-effector path (b).

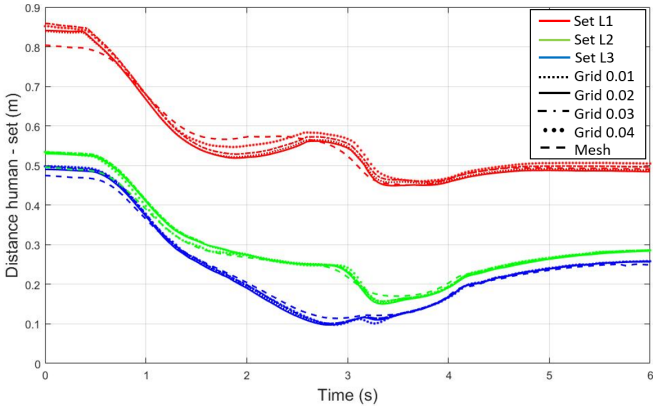

Figure 10. Minimum distances of each link set from the human.

VI. CONCLUSIONS

In this work, a practical set-up to perform fast on-line collision avoidance by means of point cloud was presented. It was shown that using relatively cheap hardware and useable software, it is possible to run the whole algorithm for human detection and collision avoidance at a frequency above 30 Hz. Calculations are made in Matlab, properly combining custom functions and built in tools; thus, the work is easily replicable and can be useful to the preliminary study of collision avoidance with dynamic objects. A comparison between two different human tracking approaches was discussed, to verify the algorithm and to highlight different robot behaviours if a convex hull of the human rather than his point cloud is utilized

for the collision avoidance. The layout with two Kinect sensors filled human self-occlusions, but the absence of a physical robot raises issues about possible robot-human occlusions: if a real Cobot is used, this could affect the point cloud reconstruction, but the approach presented in this work can be extended to bigger multi-sensor lay-outs to prevent tracking errors. Future work will focus on the real-time software implementation with a physical robot.

REFERENCES

[1] M.Melchiorre, L. Scimmi, S.Mauro, S.Pastorelli "Influence of human limb motion speed in a collaborative hand over task," in *Proceedings of the 15th International Conference on Informatics in Control, Automation and Robotics*, vol.2, pp 349-356, July 2018.

[2] A. Cherubini, R. Passama, A. Crosnier, A. Lasnier, P. Fraisse, "Collaborative manufacturing with physical human-robot interaction," *Robotics and Computer-Integrated Manufacturing*, 40, pp. 1-13, Jan. 2016.

[3] J. Shotton, A. Fitzgibbon, M.Cook, T. Sharp, M. Finocchio, R. Moore et al., "Real-time human pose recognition in parts from single depth images," in: Cipolla R., Battiato S., Farinella G. (eds) *Machine Learning for Computer Vision. Studies in Computational Intelligence*, vol. 411, Springer, Berlin, Heidelberg.

[4] Kinect SDK. https://developer.microsoft.com/it-IT/windows/kinect.

[5] J. M. D. Barros, F. Garcia, D. Sidibè, "Real-time human pose estimation from body-scanned point clouds," *International Conference on Computer Vision Theory and Applications*, Berlin, Germany, Mar. 2015.

[6] K. Y. Yeung, T. H. Kwok and C. L. Wang, "Improved skeleton tracking by a duplex kinects: a practical approach for real-time applications," In *Journal of Computing and Information Science in Engineering*, 13(4), 2013.

[7] L. Shuai, C. Li, X. Guo, B. Prabhakaran and J. Chai, "Motion capture with ellipsoidal skeleton using multiple depth cameras," in *IEEE Transactions on Visualization and Computer Graphics*, vol.23(2), Feb. 2017.

[8] L. Yang, L. Zhang, H. Dong, A. Alelaiwi and A. El Saddik, "Evaluating and improving the depth accuracy of Kinect for windows v2," *IEEE Sensors Journal*, vol. 15, pp.4275-4285, Mar. 2015.

[9] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," in *Int. J. of Robotics Research*, 5(1), pp. 98-98, 1986.

[10] M. Fischer and D. Henrich, "3D collision detection for industrial robots and unknown obstacles using multiple depth images," in *German Workshop on Robotics*, Germany, June 9-10 2009.

[11] K. B. Kaldestad, S. Haddadin, R. Belder, G. Hovland, D. A. Anisi, "Collision avoidance with potential fields based on parallel processing of 3D-point cloud data on the GPU," *IEEE Conference on Robotics and Automation*, Hong Kong, China, Jun. 2014.

[12] B. Schmidt, L. Wang, "Depth camera based collision avoidance via active robot control," in *Journal of Manufacturing Systems*, vol.33, pp. 711-718, 2014.

[13] F. Flacco, A. De Luca, "Real-time computation of distance to dynamic obstacles with multiple depth sensors", IEEE Robotics and Automation Letters, vol.2(1), Jan. 2017.

[14] J. H. Chen and K. T. Song, "Collision free motion planning for human robot collaborative safety under cartesian constraint*," IEEE International Conference on Robotics and Automation*, Brisbane, Australia, May 2018.

[15] Human tracking for multi-Kinect systems around a robot on ROS. https://github.com/kuka-isir/kinects_human_tracking.

[16] S.Mauro, L. S. Scimmi and S. Pastorelli, "Collision Avoidance Ststem for Collaborative Robotics" in Mechanisms and Machine Science, vol. 49 , pp.344-352, 2018.

[17] S. Mauro, S. Pastorelli and L. S. Scimmi, "Collision avoidance algorithm for collaborative robotics," in Int J. Of Automation Technology, vol. 11(3), pp. 481-489, 2018.

[18] L. S. Scimmi, M. Melchiorre, S. Mauro, S. Pastorelli, "Multiple collision avoidance between human limbs and robot links algorithm in collaborative tasks," in Proceedings of the 15th International Conference on Informatics in Control, Automation and Robotics, vol.2, 2018.

[19] K. Bartlett, "Jtcp(actionStr,varargin)," https://it.mathworks.com/matlabcentral/fileexchange/24524-jtcp-actionstr-varargin.

[20] R. Thomson, "TCP/IP socket communications in MATLAB using java classes", https://it.mathworks.com/matlabcentral/fileexchange/25249-tcp-ip-socket-communications-in-matlab-using-java-classes.

[21] P.Sturm, "Pinhole camera model," in Computer Vision, Springer, pp. 610-613, 2014.

[22] J. R. Terven, "Kinect 2 interface for Matlab," https://it.mathworks.com/matlabcentral/fileexchange/53439-kinect-2-interface-for-matlab.

[23] D.Frisch, "point2trimesh() - distance between point and triangulated surface." https://it.mathworks.com/matlabcentral/fileexchange/52882-point2trimesh-distance-between-point-and-triangulated-surface.

[24] F. Flacco, T. Kroger, A. De Luca, Oussama Khatib, "A depth space approach to human-robot collision avoidance," in *IEEE International Conference on Robotics and Automation,* USA, May 14, 2012.

[25] P. I. Corke, "Robotics, Vision and Control: Fundamental Algorithms in Matlab," *Springer*, London, 2017.

[26] Realtime URDF filter http://github.com/blodow/realtime_urdf_filter.